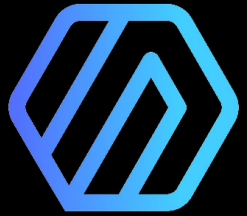
Scripting Radare2

In Javascript

–pancake

Who Am I

- Author and lead maintainer of radare2
  - Free software and security entusiast
- Mobile Security Research Analyst at NowSecure
  - Saving the world from insecure mobile apps
- Contact me at @pancake@infosec.exchange



Workshop Plan

Understand the basics of the commandline use of radare2

- Creating macros and the limitations

Learn the basics of scripting

- r2pipe, r2papi and native APIs
- Python and Javascript

Automate actions inside the shell

- Scripting in batch

The Toolchain

Components

Radare2 is composed by:

- **Tools** (rabin2, rasm2, rafind2, ..)
- **Libraries** (r_cons, r_util, r_core, ..)
- **Plugins** (arch_arm, debug_gdb, ..)

Tools

- Rax2 - Calculator / Converter
- Rasm2 - Assembler / Disassembler
- Rabin2 - Binary Header Parser
- Rahash2 - Checksum and Crypto
- Rafind2 - Search / Carve / Scan for patterns
- Radiff2 - Find difference between two files
- Rarun2 - Run programs with custom
- Rasign2 - Generate Binary Signatures
- Radare2 - Everything in a single place

One For All

Radare2, also known as “r2”.

- Links with all the r2 libraries
- REPL with mnemonic commands
- Support various Visual modes
- Entrypoint for the whole toolchain
- Scripting, plugins

Introduction to the CLI

Basic Commands

s = seek (s 0x/s..)

p = print (px / pd)

f = flags

i = info

w = write

q = quit

a = analysis

V = visual/panels

e = eval config

? = help/math

! = system shell

d = debugger

Command Operators

| = redirect to process (like in posix shell)

> = redirect to file ($file are internal) see 2> and >>

~ = internal grep (indent json, xml, code, filter words)

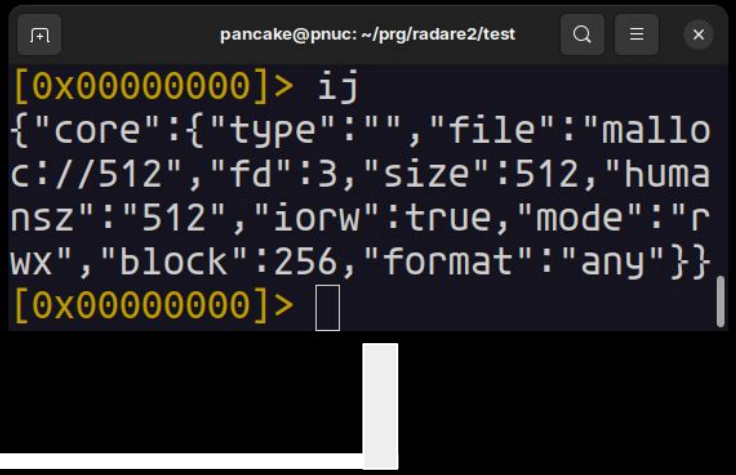
= comment

; = command separator

? = show command help

Command Suffixes

- **?** = help message
- ***** = r2 commands
- **q** = quiet
- **,** = comma separated values
- **k** = key-value
- **j** = json



Command Prefixes

- (**number**) = repeat a command N times
- **'** = single quote, to avoid parsing special characters
- **?t** = calculate execution time
- **:** = io command
- **``** = replace command output inline
- **.** = run script

Command Iterators

Useful to run commands in different items

- Functions, flags, registers, symbols, basic blocks, ..

- `@` - temporal seek
- `@@` - repeat command on different places
- `@@@` - advanced repeat actions

See `@?` `@@?` `@@@?` for help

Useful Commands

Combine and learn new commands every day!

- Recursive Help: `?*`
- JSON indent (json path queries like jq): `~{}`
- HUD filtering: `~…`
- Analyse all symbols: `af @@ sym*`
- Set, list flags: `f`
- Comments: `CC`

Exercise

Practice few commands in the shell and see the different representations for the output we can get from them

Scripting with Commands

Running Scripts

We know how to use the shell.

- `r2 -i` or the `.` command.

What about running a command and capturing the output displayed in return?

- **That's what we call "r2pipe"**

We can also use bindings to the native API (rlang)

JSON output

Appending j to the command we get the output in json format, so its easy to process the output from r2, python, jq or js.

- The internal grep permits json path filtering ~{}
- We can script r2 with shellscripts and jq
- r2p command can run

Environment

The % command allows us to manage the environment variables inside the r2 process.

- Shellscripts can use the R2_ envs to take some info
- R2PIPE filedescriptors are exposed as env vars too

Macro Command

The (command defines small expressions to implement loops, conditionals, argument replacement or call other macros.

- Syntax is a bit cryptic, but can be handy for oneliners.

Definitively we need a *real language* if we want something larger

#!C

Scripting in C

The Rlang plugin for C compiles the given C code into a shared library that takes the RCore instance as argument.\

So you can run native code using the native APIs directly

```
> #!c hello.c
```

#!Pipe

Scripting via #!pipe

Exposes two filedescriptors in environment variables to write a command and read the output in the other.

- r2p program can be used from shellscripts
- All r2pipe implementations support this
- We have access to the environment

Exercise

Write a shellscript that interacts with r2 using r2p and jq

R2Pipe

Hello Pipe!

Example using the basic r2pipe api

```
import r2pipe

r2 = r2pipe.open("/bin/ls")
out = r2.cmd("?E Hello World")
print(out)
r2.quit()
```

Backends

R2Pipe can be used in different environments:

- Spawn + pipes
- Spawn + stdio
- Fork current session + pipes (#!pipe)
- Talking to an HTTP websever /cmd
- Dlopen RCore API

Supported Languages

Check r2pipe and rlang repositories

- C
- C++
- C#
- D
- F#
- Vala
- Guile
- LUA
- Ruby
- Python
- Perl
- Go
- Haskell
- NewLisp
- REXX
- Swift
- TypeScript
- V
- Wasm
- Wren
- Crystal
- Nim
- PHP
- Prolog
- Rust
- Bash
- Clojure
- Java
- Vala
- Zig
- Erlang
- Ocaml
- Lisp
- Poke
- AWK
- TCL
- Scheme
- JavaScript
- Assembly
- Kotlin

JSON with cmdj()

Appending `j` to any command in r2 shows JSON.

Using the `cmdj` methods returns an object.

We can autogenerate object schemas and have autocompletion in our favourite editor!

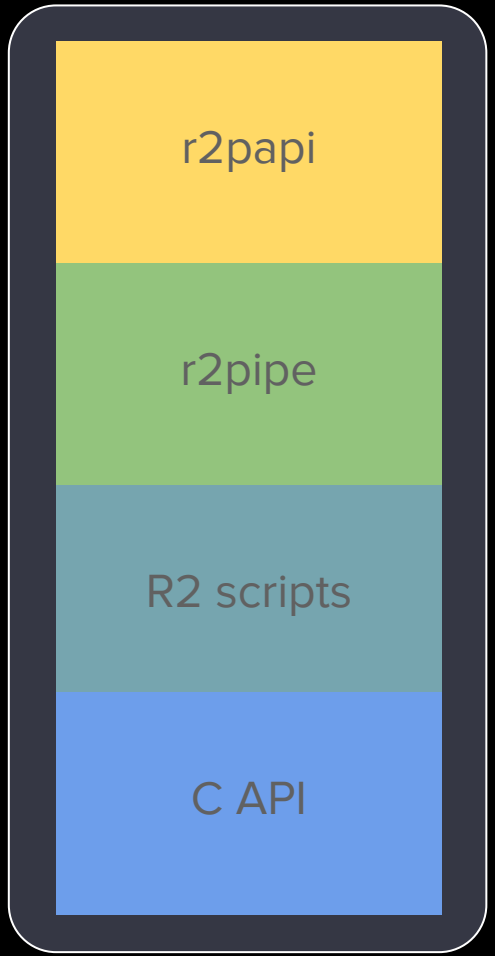
```
cmdj(command: string) : string {
  return JSON.parse(this.cmd(command));
}
```

Performance

Who said speed?

Sometimes we don't need the output

- Use **`cmd0`** or **`call0`** commands



Cmd vs Call

Running a command implies too much internal work sometimes that we can bypass with .call()

- Don't parse special characters
- Avoid command injection
- Support temporal seek .callAt()
- Faster execution for large scripts

R2Skel

Repository containing sample plugins, scripts to use as skeleton for new projects.

- `r2pm -ci r2skel`
- `r2pm -r r2skel`
  - Use -l and -L flags to list supported languages and templates available
- `r2pm -r r2skel r2-plugin-core-ts newplugin`
  - Create the “newplugin” directory

Exercise

Try r2pipe for Python, spawn a new instance and run commands

R2JS

Why Javascript

Widely available, easy to learn and use. No setjmp.
Many languages have it as a target for transpilation.

- Nim, TypeScript, V, Scala, Dart, LUA, Scheme,...

We ship quickjs, scripts must be named `.r2.js`

Exercise

Use r2 -j and check the example scripts shipped with r2 to run them

R2Papi

R2Papi

Stands for R2Pipe API.

- Idiomatic API on top of r2pipe
- Relies on commands to work
- Clean and simple API with your favourite IDE
- Typescript and Python
- Frida-like APIs for reusing knowledge

r2papi

What about having an idiomatic and high level API on top of the r2pipe primitive?

- Similar to the Frida API (NativePointer, ..)

```
}
/**
 * Copy N bytes from current pointer to the destination
 *
 * @param {string|NativePointer|number} destination address
 * @param {string|number} amount of bytes
 */
async copyTo(addr: string|NativePointer|number, size: string|number) : Promise<void> {
    this.api.call(`wf ${this.addr} ${size} @ ${addr}`)
}
```

Exercise

Write a simple structured data parser using the NativePointer API

R2Pipe2

R2Pipe2

Introduced in r2-5.9.x, still WIP, and not fully handled; needs more testing, feedback and contributions.

- Protocol is there
- Fully compatible with r2pipe
- Uses the `{` command from r2
- Captures stderr and return code and value

Questions?